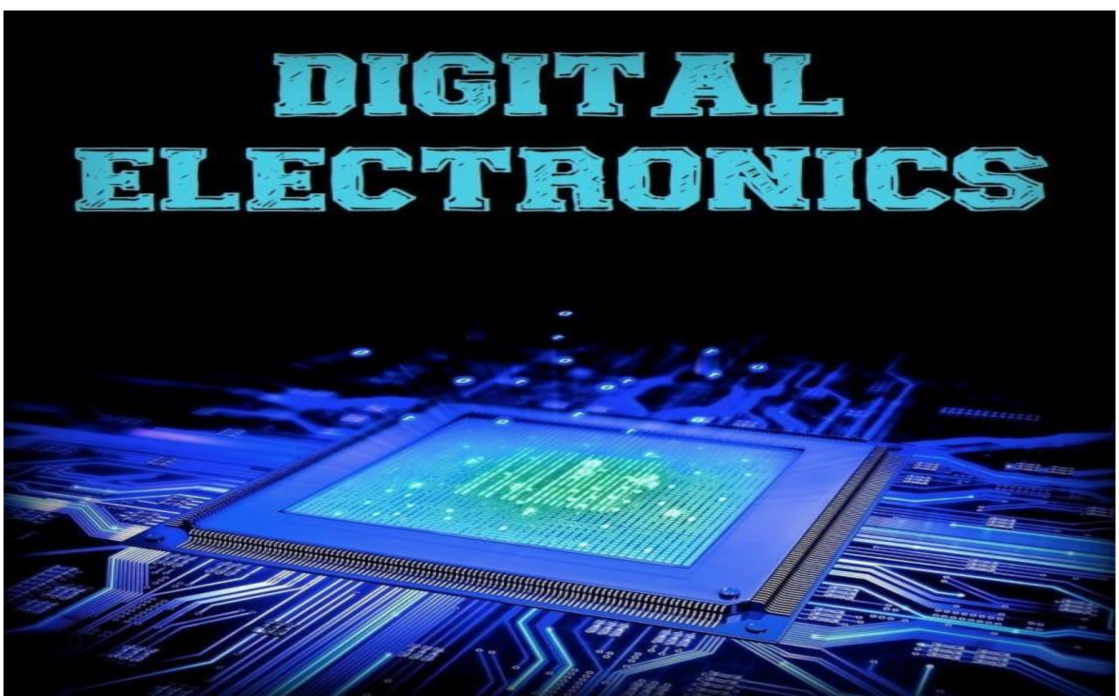
Abanob Evram

Assignmen1[Extra]



[Q1]

1)

Design a Verilog module using **generate conditional construct** that takes a 3-bit binary input (A[2:0]) and produces a corresponding encoded output (B[6:0]). The encoding method can be selected using a parameter called USE_GRAY, which can be set to either 1 or 0. If USE_GRAY is set to 1, the module should perform Gray encoding on the input using always block. If USE_GRAY is set to 0, the module should perform one-hot encoding on the input using always block. Default value for USE_GRAY = 1.

One-hot encoding is a way of representing data in a binary string in which only a single bit can be 1, while all others are 0.

Gray encoding, also known as Gray code, is a binary numeral system where consecutive values differ by only one bit. In Gray encoding, each binary number is represented such that only one bit changes from one value to the next, reducing the possibility of errors during transitions.

Decimal	Binary	Gray code	One-hot
0	000	000	0000000
1	001	001	0000001
2	010	011	0000010
3	011	010	0000100
4	100	110	0001000
5	101	111	0010000
6	110	101	0100000
7	111	100	1000000

Write **two testbenches** to verify the functionality of the design in both parameter values. The first testbench should instantiate the design and override the parameter USE_GRAY to be equal 1 and the second testbench to override the USE_GRAY to be equal 0. Both testbenches should be exhaustive testbenches and check the results from the waveforms.

The design code:

```
module encoder(A,B);
parameter USE_GRAY=0;
input [2:0] A;
output reg [6:0] B;
 generate
   if (USE_GRAY==0)
      always @(A)
      case(A)
         0:B=0;
         1:B=1;
         2:B=2;
         3:B=4;
         4:B=8;
         5:B=16;
         6:B=32;
         7:B=64;
       endcase
   else
      always @(A)
      case(A)
         0:B=0;
         1:B=1;
         2:B=3;
         3:B=2;
         4:B=6;
         5:B=7;
         6:B=5;
         7:B=4;
       endcase
 endgenerate
endmodule
```

The testbench code for gray:

```
module encoder_gray_tb();
parameter USE_GRAY_tb=1;
reg [2:0] A_tb;
reg [6:0] B_expected;
wire [6:0] B_dut;
```

```
encoder #(USE_GRAY_tb) dut(A_tb,B_dut);
integer i ;
 initial begin

    for(i=0;i<8;i=i+1)    begin
       A_tb=i;
        case(A_tb)
           0:B_expected=0;
           1:B_expected=1;
           2:B_expected=3;
           3:B_expected=2;
           4:B_expected=6;
           5:B_expected=7;
           6:B_expected=5;
           7:B_expected=4;
        endcase
        #10
        if (B_expected!=B_dut) begin
        $display("Errror....");
        $stop;
        end
    end
   $stop;
   end
  initial begin
  $monitor("A_tb=%d,B_expected=%b",A_tb,B_expected);
  end
endmodule
```

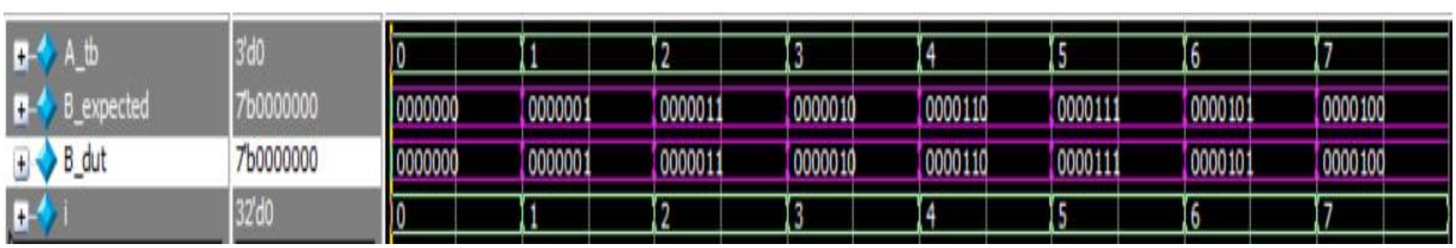
The testbench code for one hot:

```
module encoder_one_hot_tb();
parameter USE_GRAY_tb=0;
reg [2:0] A_tb ;
reg [6:0] B_expected;
wire  [6:0] B_dut;
encoder #(USE_GRAY_tb) dut(A_tb,B_dut);
integer i ;
 initial begin

    for(i=0;i<8;i=i+1)    begin
```

```
        A_tb = i;
         case(A_tb)
           0:B_expected=0;
           1:B_expected=1;
           2:B_expected=2;
           3:B_expected=4;
           4:B_expected=8;
           5:B_expected=16;
           6:B_expected=32;
           7:B_expected=64;
         endcase
         #10
         if (B_expected!=B_dut) begin
         $display("Errror....");
         $stop;
         end
      end
     $stop;
     end
    initial begin
    $monitor("A_tb=%d,B_expected=%b",A_tb,B_expected);
    end
  endmodule
```

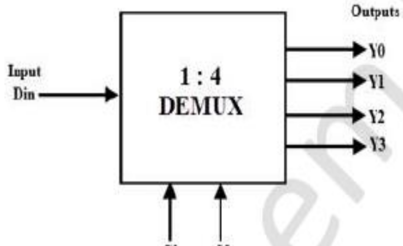
Wave of Gray_code :-

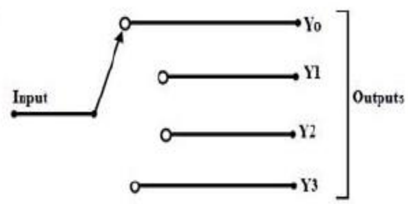


Wave of One_hot :-

2) Design a 1-to-4 demultiplexer (Demux) using Verilog. The demultiplexer should have a single input (D) and two select inputs (S[1:0]). The output should consist of four signals (Y[3:0]), where the input signal (D) is routed to one of the four outputs based on the select inputs. Write the Verilog code for this demultiplexer and simulate its functionality using an exhaustive self-checking testbench.

Data Input	Select Inputs		Outputs			
D	S_1	S_0	Y_3	Y_2	Y_1	Y_0
D	0	0	0	0	0	D
D	0	1	0	0	D	0
D	1	0	0	D	0	0
D	1	1	D	0	0	0





[Q2]

The design code:

```
module Demux(D,S,Y);
input D;
input [1:0] S;
output reg [3:0] Y;
always @(*) begin
case(S)
0:Y={3'b000,D};
1:Y={2'b00,D,1'b0};
2:Y={1'b0,D,2'b00};
3:Y={D,3'b000};
endcase
end
endmodule
```

The testbench code:

```
module Demux_tb();
reg D_tb;
reg [1:0] S_tb;
reg [3:0] Y_excpected;
wire [3:0] Y_dut;
Demux dut(D_tb,S_tb,Y_dut);
integer i,j;
initial begin
  for(i=0;i<2;i=i+1) begin
D_tb=i;
  for(j=0;j<4;j=j+1)begin
S_tb=j;
case(S_tb)
0:Y_excpected={3'b000,D_tb};
1:Y_excpected={2'b00,D_tb,1'b0};
2:Y_excpected={1'b0,D_tb,2'b00};
3:Y_excpected={D_tb,3'b000};
```

```
endcase
#10
if(Y_excpected!=Y_dut)begin
$display("Errror");
$stop;
end
  end
  end
$stop;
end
initial begin
$monitor("D_tb=%d , Y_excpected=%b",D_tb,Y_excpected);
end
endmodule
```